Fault Tolerance

Distributed Software Systems

Definitions

- **Availability**: probability the system operates correctly at any given moment
- **Reliability**: ability to run correctly for a long interval of time
- **Safety**: failure to operate correctly does not lead to catastrophic failures
- **Maintainability**: ability to "easily" repair a failed system

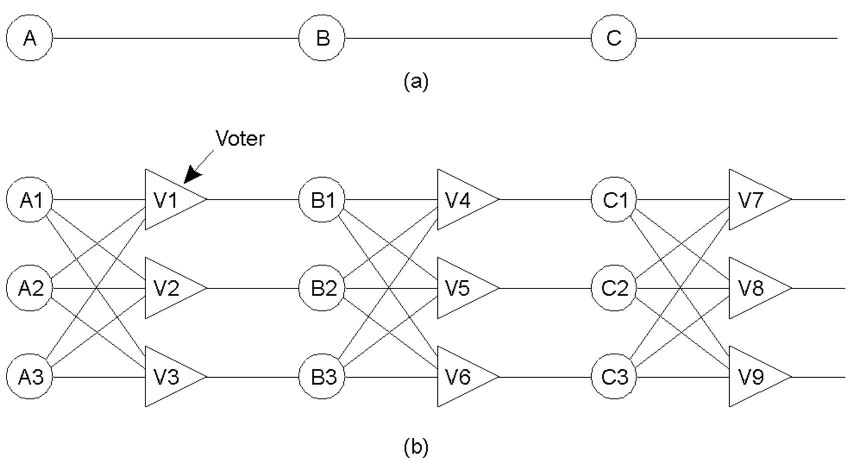
Failure Models

Different types of failures.

Type of failure	Description
Crash failure	A server halts, but is working correctly until it halts
Omission failure *Receive omission* *Send omission*	A server fails to respond to incoming requests A server fails to receive incoming messages A server fails to send messages
Timing failure	A server's response lies outside the specified time interval
Response failure *Value failure* *State transition failure*	The server's response is incorrect The value of the response is wrong The server deviates from the correct flow of control
Arbitrary failure	A server may produce arbitrary responses at arbitrary times

Failure Masking by Redundancy

Triple modular redundancy.

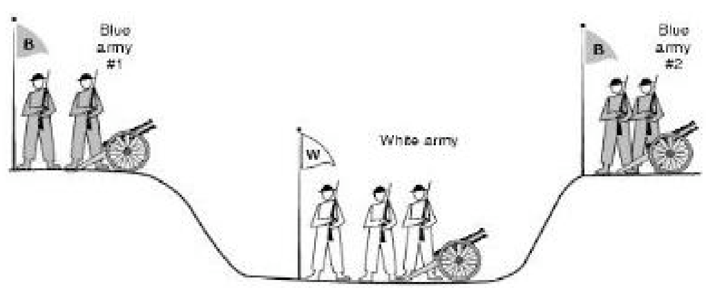


Agreement in Faulty Systems

- Many things can go wrong…
- Communication
  - Message transmission can be unreliable
  - Time taken to deliver a message is unbounded
  - Adversary can intercept messages
- Processes
  - Can fail or team up to produce wrong results
- Agreement very hard, sometime impossible, to achieve!

Two-Army Problem

- "Two blue armies need to simultaneously attack the white army to win; otherwise they will be defeated. The blue army can communicate only across the area controlled by the white army which can intercept the messengers."



- What is the solution?

Byzantine Agreement [Lamport et al. (1982)]

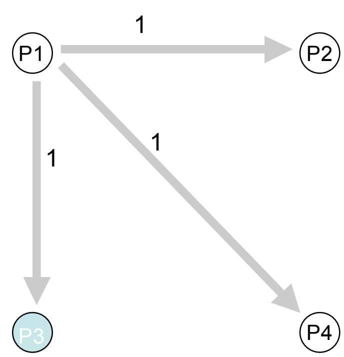
- Goal:
  - Each process learn the true values sent by correct processes
- Assumptions:
  - Every message that is sent is delivered correctly
  - The receiver knows who sent the message
  - Message delivery time is bounded

Byzantine Agreement Result

- In a system with m faulty processes agreement can be achieved only if there are $2m+1$ functioning correctly
- Note: This result only guarantees that each process receives the true values sent by correct processors, but it does not identify the correct processes!

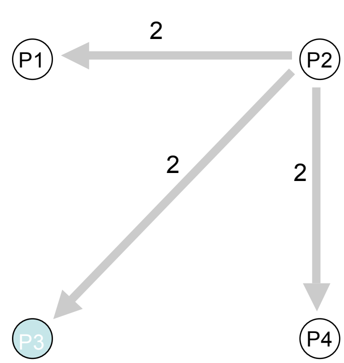
Byzantine General Problem: Example

- Phase 1: Generals announce their troop strengths to each other



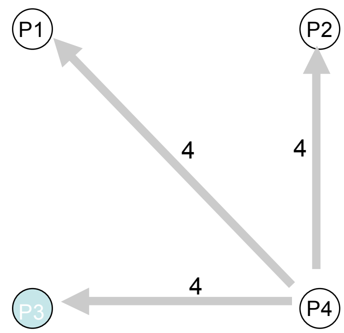
Byzantine General Problem: Example

- Phase 1: Generals announce their troop strengths to each other



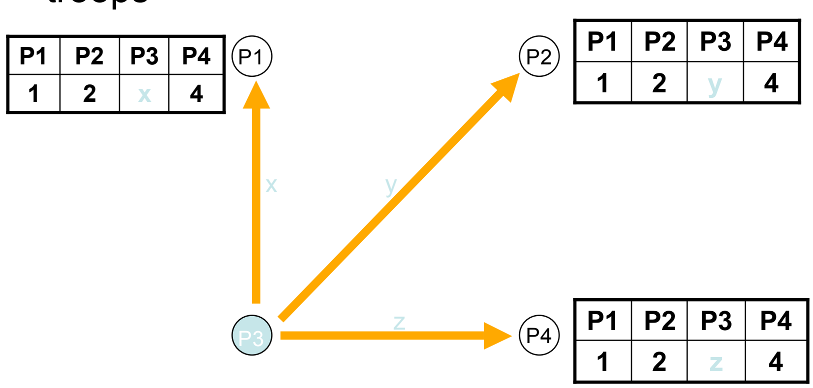
Byzantine General Problem: Example

- Phase 1: Generals announce their troop strengths to each other



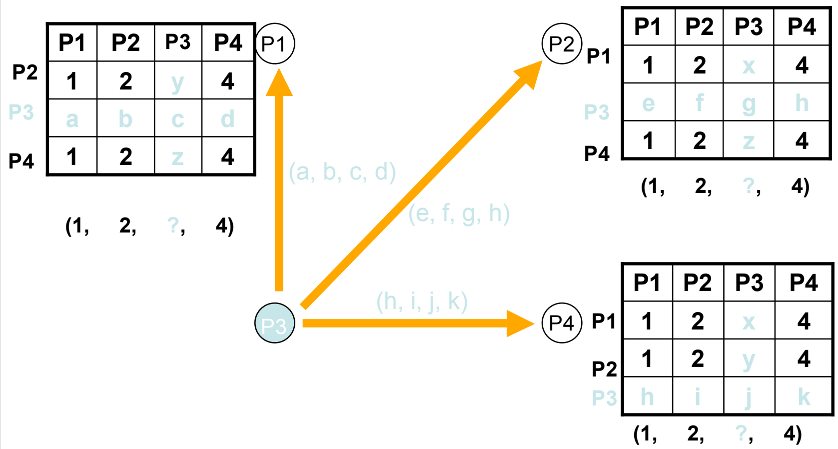
Byzantine General Problem: Example

- Phase 2: Each general construct a vector with all troops



Byzantine General Problem: Example

- Phase 3: Generals send their vectors to each other and compute majority voting



P1:

	P1	P2	P3	P4
P2	1	2	y	4
P3	a	b	c	d
P4	1	2	z	4

(1, 2, ?, 4)

P2:

	P1	P2	P3	P4
P1	1	2	x	4
P3	e	f	g	h
P4	1	2	z	4

(1, 2, ?, 4)

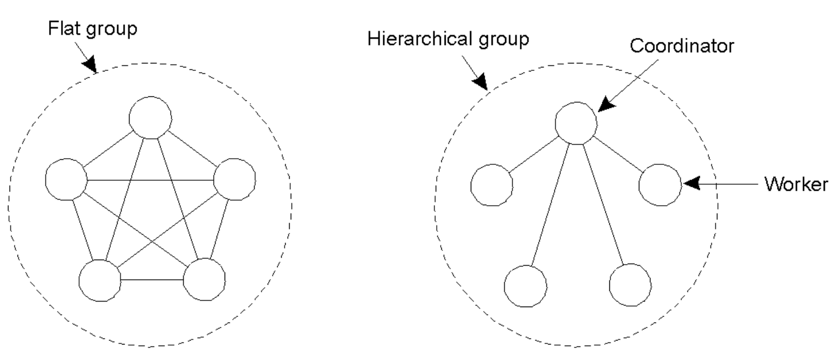
P4:

	P1	P2	P3	P4
P1	1	2	x	4
P2	1	2	y	4
P3	h	i	j	k

(1, 2, ?, 4)

Flat Groups versus Hierarchical Groups

a) Communication in a flat group.
b) Communication in a simple hierarchical group

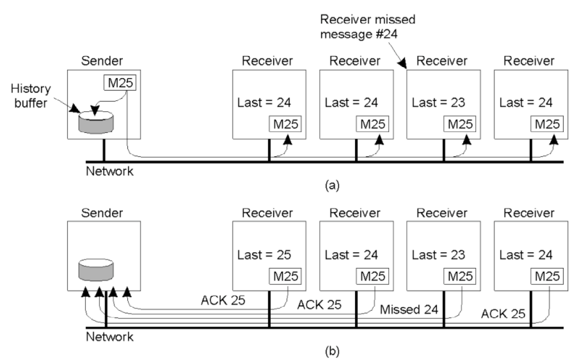


(a) (b)

Reliable Group Communication

- **Reliable multicast**: all nonfaulty processes which do not join/leave during communication receive the message
- **Atomic multicast**: all messages are delivered in the same order to all processes

Basic Reliable-Multicasting Schemes

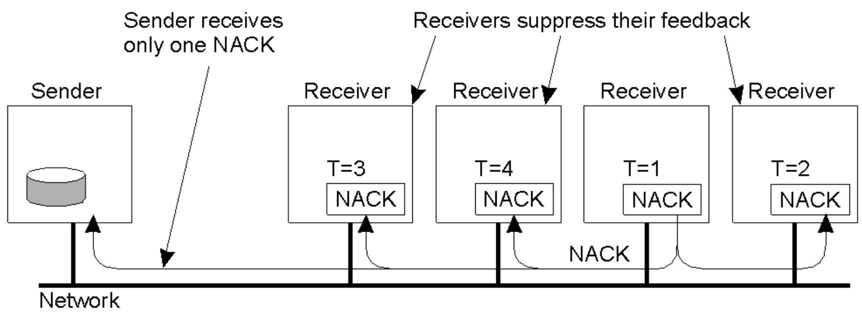


A simple solution to reliable multicasting when all receivers are known and are assumed not to fail

a) Message transmission

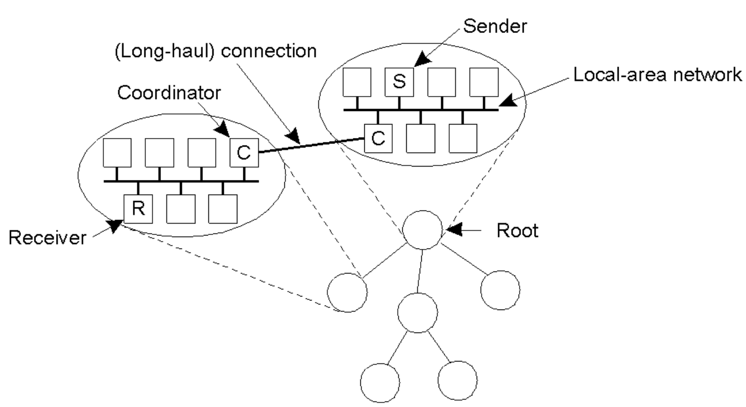
b) Reporting feedback

Nonhierarchical Feedback Control



- Several receivers have scheduled a request for retransmission, but the first retransmission request leads to the suppression of others.

Hierarchical Feedback Control



The essence of hierarchical reliable multicasting.

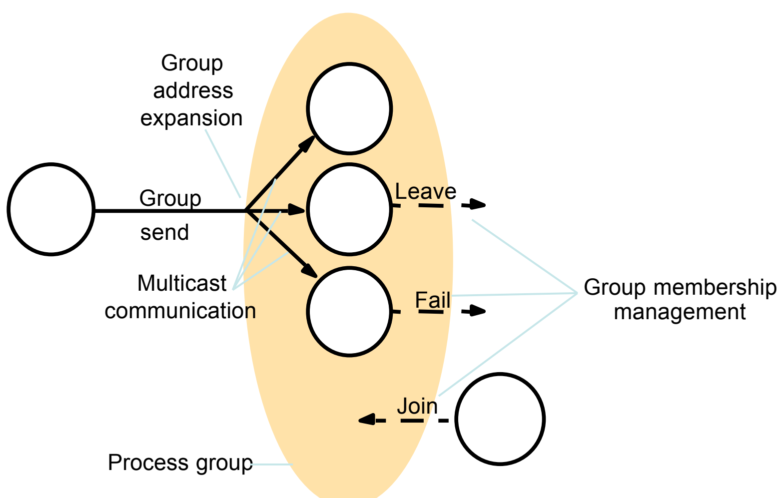
a) Each local coordinator forwards the message to its children.

b) A local coordinator handles retransmission requests.

Group communication

- Role of group membership service
  - Provide an interface for group membership changes
  - Implement a failure detector
  - Notify members of group membership changes
  - Perform group address expansion

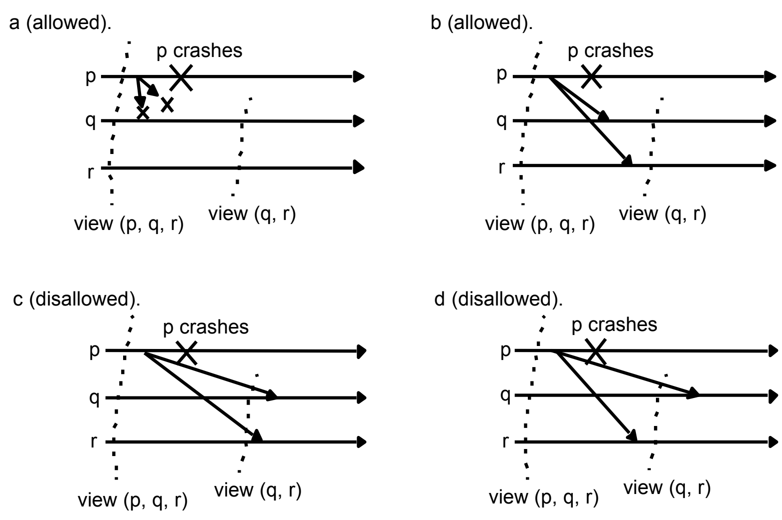
Services provided for process groups



View delivery

- A view reflects current membership of group
- A view is delivered when a membership change occurs and the application is notified of the change
  - Receiving a view is different from delivering a view
    - All members have to agree to the delivery of a view
- View-synchronous group communication
  - the delivery of a new view draws a conceptual line across the system and every message is either delivered on side or the other of that line

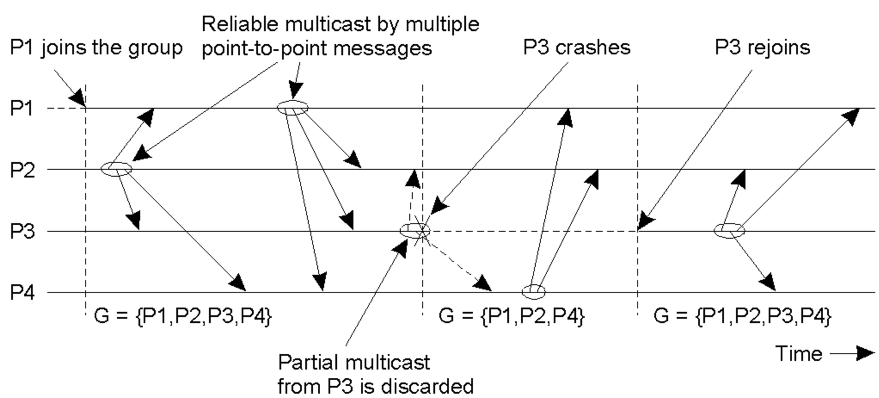
View-synchronous group communication



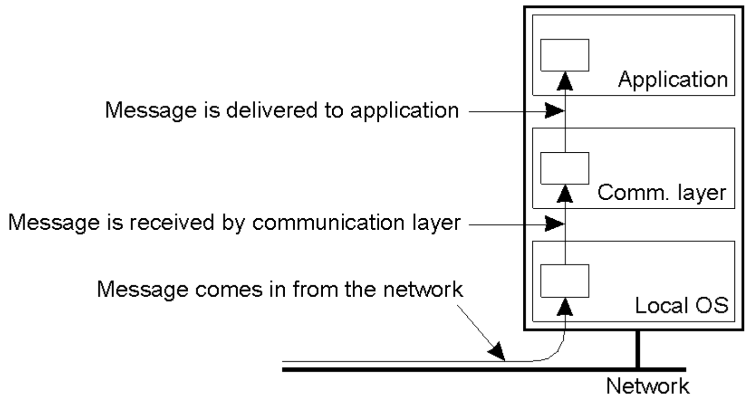
Atomic Multicast

- All messages are delivered in the same order to “all” processes
- **Group view**: the set of processes known by the sender when it multicast the message
- **Virtual synchronous multicast**: a message multicast to a group view G is delivered to all nonfaulty processes in G
  - If sender fails after sending the message, the message may be delivered to no one

Virtual Synchronous Multicast



Virtual Synchrony Implementation [Birman et al 1991]



- The logical organization of a distributed system to distinguish between message receipt and message delivery

Virtual Synchrony Implementation: [Birman et al., 1991]

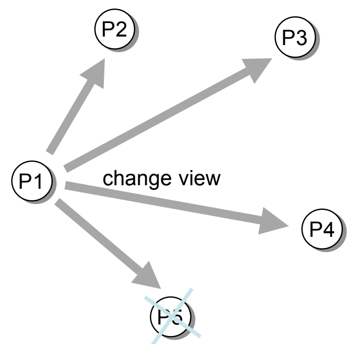
- Only stable messages are delivered

- **Stable message**: a message received by all processes in the message's group view

- Assumptions (can be ensured by using TCP):
  - Point-to-point communication is reliable
  - Point-to-point communication ensures FIFO-ordering

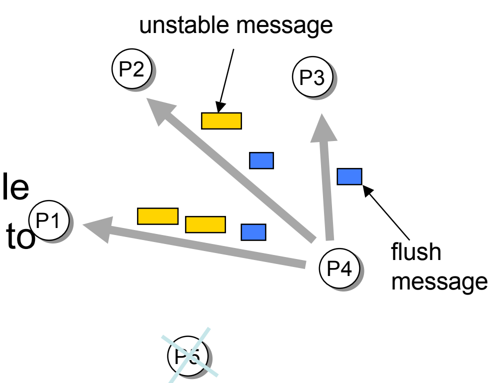
Virtual Synchrony Implementation: Example

- G_i = {P1, P2, P3, P4, P5}
- P5 fails
- P1 detects that P5 has failed
- P1 send a "view change" message to every process in G_{i+1} = {P1, P2, P3, P4}



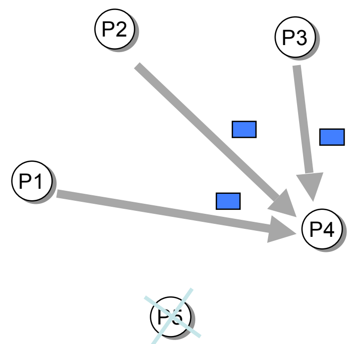
Virtual Synchrony Implementation: Example

- Every process
  - Send each unstable message m from G_i to members in G_{i+1}
  - Marks *m* as being stable
  - Send a flush message to mark that all unstable messages have been sent



Virtual Synchrony Implementation: Example

- Every process
  - After receiving a flush message from any process in G_{i+1} installs G_{i+1}



Message Ordering

- Discussed last week how we can implement an ordered multicast
  - **FIFO-order**: messages from the same process are delivered in the same order they were sent
  - **Causal-order**: potential causality between different messages is preserved
  - **Total-order**: all processes receive messages in the same order
- Total ordering does not imply causality or FIFO!
- Atomicity is orthogonal to ordering

Message Ordering and Atomicity

Multicast	**Basic Message Ordering**	**Total-ordered Delivery?**
Reliable multicast	None	No
FIFO multicast	FIFO-ordered delivery	No
Causal multicast	Causal-ordered delivery	No
Atomic multicast	None	Yes
FIFO atomic multicast	FIFO-ordered delivery	Yes
Causal atomic multicast	Causal-ordered delivery	Yes